EXPERIMENT NO: 02

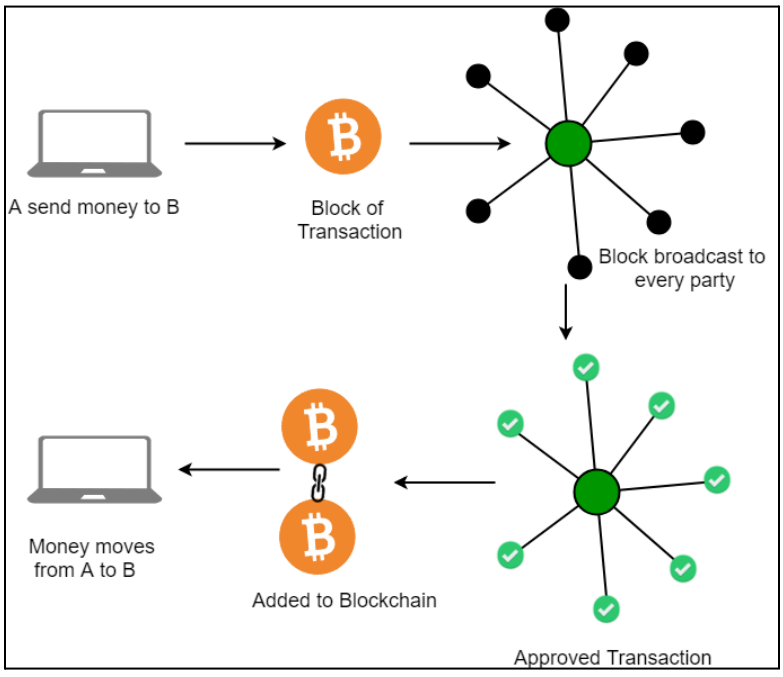
AIM: Create a Blockchain using Python

THEORY:

1. What is blockchain

Blockchain is a decentralized digital ledger technology that securely records transactions across multiple computers in a way that prevents alteration. Each transaction is grouped into a block, which is then linked to the previous block using cryptographic hashes, forming a chain. This structure ensures that once data is added, it becomes nearly impossible to change or delete, providing a permanent and transparent record.

1. **Decentralization:** No single entity controls the entire blockchain; instead, it is maintained by a network of participants (nodes).
2. **Transparency:** All transactions are visible to participants, fostering trust and accountability.
3. **Immutability:** Once a block is added to the chain, altering any previous block requires consensus from the majority of the network, making fraud extremely difficult.



2. What is block

A block in a blockchain network is similar to a link in a chain. In the field of cryptocurrency, blocks are like records that store transactions like a record book, and those are encrypted into a hash tree. There are a huge number of transactions occurring every day in the world. The users need to keep track of those transactions, and they do it with the help of a block structure. The block structure of the blockchain is mentioned in the very first diagram in this article.

Components of block

1. Header: It is used to identify the particular block in the entire blockchain. It handles all blocks in the blockchain. A block header is hashed periodically by miners changing the nonce value as part of normal mining activity, also Three sets of block metadata are contained in the block header.
2. Block Address/ Hash: It is used to connect the i+1th block to the ith block using the hash. In short, it is a reference to the hash of the previous (parent) block in the chain.
3. Timestamp: It is a system that verifies the data into the block and assigns a time or date of creation for digital documents. The timestamp is a string of characters that uniquely identifies the document or event and indicates when it was created.
4. Nonce: A nonce number which used only once. It is a central part of the proof of work in the block. It is compared to the live target if it is smaller or equal to the current target. People who mine, test, and eliminate many Nonce per second until they find that Valuable Nonce is valid.
5. Merkel Root: It is a type of data structure frame of different blocks of data. A Merkle Tree stores all the transactions in a block by producing a digital fingerprint of the entire transaction. It allows the users to verify whether a transaction can be included in a block or not

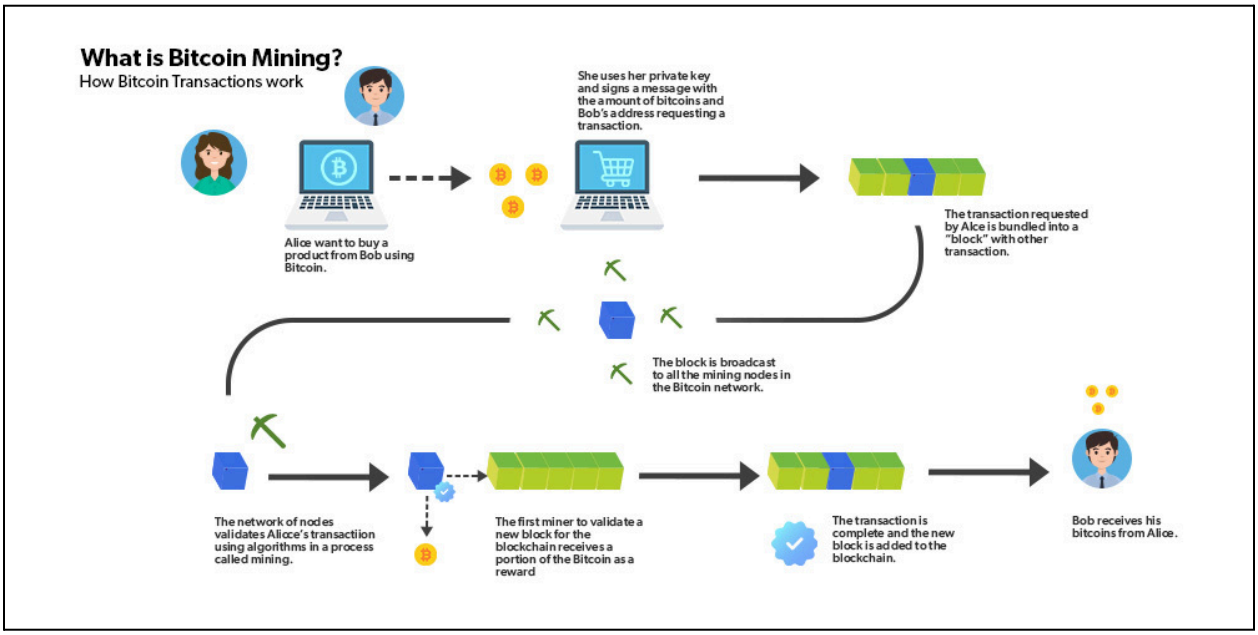
Example:

Block Number: 105678
Block Header:

- Previous Block Hash: 0000000000000000000a3f2c9e7b1d45c8fa9a1e7b6d9c2f0a4e3b1c8d9f
- Merkle Root:4f3c2a1b9e8d7c6b5a4f3e2d1c9b8a7f6e5d4c3b2a1f9e8d7c6b5a4
- Timestamp:2026-01-29 14:32:10 UTC
- Nonce: 845932
- Hash (Block Address): 00000000000000000005d6e8f3c2a9b1e7d4f6c8a2e9b5d3f1c7…
- Transactions:

1. Alice → Bob : 0.5 BTC
2. Charlie → Dave : 1.2 BTC
3. Eve → Frank : 0.3 BTC

3. Process of mining



Step 1: Collect Transactions

- All pending transactions are collected into a new block.
- Example: Alice sends 50 coins to Bob.

Step 2: Create Block Header

The block header contains:

- Previous block hash
- Transaction data
- Timestamp
- Nonce (initially set to 0)

Step 3: Proof of Work (PoW)

- The miner repeatedly changes the nonce to find a valid hash.
- The hash must satisfy the difficulty requirement.
- Difficulty: Number of leading zeroes in the hash (e.g., 4 leading zeroes).

SHA-256(block data + nonce) → 0000ab34cd...

- This process requires high computational power, hence called Proof-of-Work.

Step 4: Validate and Broadcast

- Once a valid hash is found, the block is broadcast to the network.

- Other nodes verify:
    1. Hash correctness
    2. Nonce validity
    3. Previous block hash link

Step 5: Add to Blockchain

- If verified, the block is added permanently to the blockchain.
- The miner receives a reward in cryptocurrency.

4. How to check the validity of block in blockchain?

1. Verify Block Hash

- Recalculate the hash using the block header.
- Check if it matches the stored block hash.
- Example: Calculated hash = 0000ab45cd

2. Check Proof of Work

- Ensure the hash meets the difficulty target.
- Hash must have required leading zeros.
- Example:Required: 0000xxxx → Found: 0000f9a2

3. Validate Previous Block Hash

- Confirm the previous block hash matches the hash of the last block in the chain.
- Example: Previous hash = 0000a1b2c3

4. Verify Transactions

- Check all transactions are valid.
- Ensure no double spending.

- Verify digital signatures.
  Example: Alice has sufficient balance

5. Verify Merkle Root

- Recalculate the Merkle root from transactions.
- Compare it with the Merkle root in the block header.
- Example:Merkle root matches

6. Check Timestamp

- Timestamp should be valid and not in the future.
- Example:2026-01-29 10:30 AM

7. Check Block Size & Format

- Block size and structure must follow protocol rules.

8. Validate Genesis Block

● The first block (Genesis block) is hardcoded.
● Previous hash = "0"
● Must not be altered.

Final Result

If all checks pass → Block is valid
If any check fails → Block is rejected

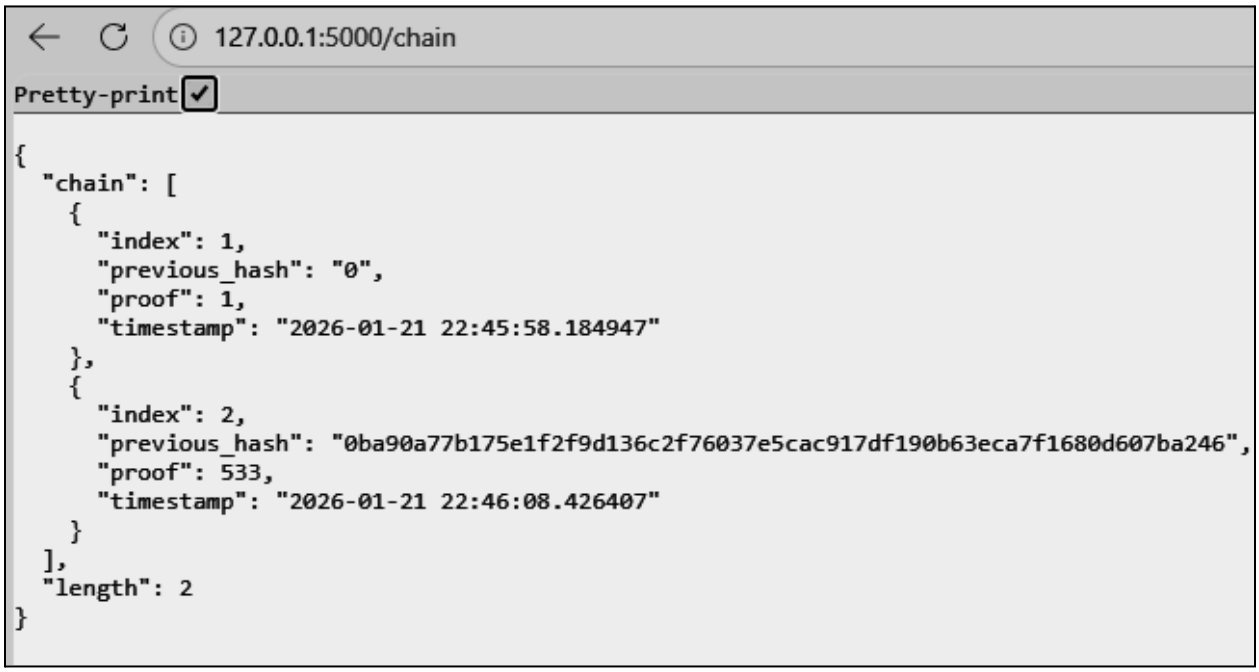
CODE:

```
import datetime
import hashlib
import json
from flask import Flask, jsonify

class Blockchain:
    def __init__(self):
        self.chain = []
        self.create_block(proof=1, previous_hash='0')
```

```
def create_block(self, proof, previous_hash):
    block = {
        'index': len(self.chain) + 1,
        'timestamp': str(datetime.datetime.now()),
        'proof': proof,
        'previous_hash': previous_hash
    }
    self.chain.append(block)
    return block

def get_previous_block(self):
    return self.chain[-1]

def proof_of_work(self, previous_proof):
    new_proof = 1
    while True:
        hash_operation = hashlib.sha256(
            str(new_proof**2 - previous_proof**2).encode()
        ).hexdigest()
        if hash_operation[:4] == '0000':
            return new_proof
        new_proof += 1

def hash(self, block):
    encoded_block = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(encoded_block).hexdigest()

def is_chain_valid(self, chain):
    previous_block = chain[0]
    index = 1

    while index < len(chain):
        block = chain[index]

        if block['previous_hash'] != self.hash(previous_block):
            return False

        hash_operation = hashlib.sha256(
            str(block['proof']**2 - previous_block['proof']**2).encode()
```

```
            ).hexdigest()

            if hash_operation[:4] != '0000':
                return False

            previous_block = block
            index += 1

        return True

app = Flask(__name__)
blockchain = Blockchain()

@app.route('/mine', methods=['GET'])
def mine():
    previous_block = blockchain.get_previous_block()
    proof = blockchain.proof_of_work(previous_block['proof'])
    previous_hash = blockchain.hash(previous_block)
    block = blockchain.create_block(proof, previous_hash)

    return jsonify({
        'message': 'Congratulations, you just mined a block!',
        'index': block['index'],
        'timestamp': block['timestamp'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash']
    }), 200

@app.route('/chain', methods=['GET'])
def chain():
    return jsonify({
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }), 200

@app.route('/validate', methods=['GET'])
def validate():
    if blockchain.is_chain_valid(blockchain.chain):
        return jsonify({'message': 'All good. The Blockchain is valid.'}), 200
    return jsonify({'message': 'Houston, we have a problem. The Blockchain is not valid.'}), 200
```

app.run(host='0.0.0.0', port=5000)




127.0.0.1:5000/mine
Pretty-print
{
"index": 3,
"message": "Congratulations, you just mined a block!",
"previous_hash": "67ba23a3ef2ecd03ae5338a32075eae0155ee96e402d9f51b7f3329869d9c45b",
"proof": 45293,
"timestamp": "2026-01-21 22:49:48.790923"
}


127.0.0.1:5000/chain
Pretty-print
{
"chain": [
{
"index": 1,
"previous_hash": "0",
"proof": 1,
"timestamp": "2026-01-21 22:45:58.184947"
},
{
"index": 2,
"previous_hash": "0ba90a77b175e1f2f9d136c2f76037e5cac917df190b63eca7f1680d607ba246",
"proof": 533,
"timestamp": "2026-01-21 22:46:08.426407"
},
{
"index": 3,
"previous_hash": "67ba23a3ef2ecd03ae5338a32075eae0155ee96e402d9f51b7f3329869d9c45b",
"proof": 45293,
"timestamp": "2026-01-21 22:49:48.790923"
}
],
"length": 3
}


127.0.0.1:5000/validate
Pretty-print
{"message":"All good. The Blockchain is valid."}

CONCLUSION:

Thus, the aim of creating a Blockchain using Python was successfully achieved. In this experiment, the basic structure of a blockchain was implemented by creating blocks containing transaction data, timestamps, hashes, nonces, and links to previous blocks. The concept of mining using the Proof-of-Work (PoW) algorithm was applied to ensure data integrity and security. Each block was validated by checking the hash value, previous block reference, and difficulty condition. This experiment helped in understanding how blockchain maintains transparency, immutability, and security while recording transactions. Hence, the working principle of blockchain technology was studied and verified using Python.